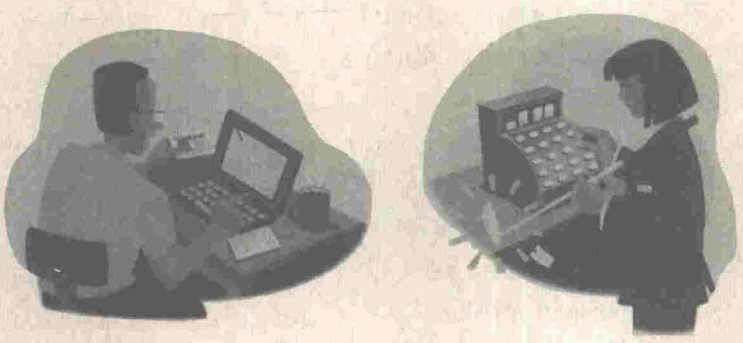
ภาษิต ภัทรมูล

PowerBuilder 5.0 : มุมมองที่แตกต่าง ตอนที่ 2



แนะนำให้รู้จักกับ Application ที่สร้างโดย PowerBuilder โดยเน้นในองค์ประกอบ เพื่อการประยุกต์ใช้งานต่อไป

ตอนที่แล้วได้พูดถึง PowerBuilder ในลักษณะ Review ตอนนี้เราจึงจะมาลงลึกถึงรายละเอียดที่มีความจำเป็นต้องรู้ เมื่อเราตกลงจะใช้ PowerBuilder ในการพัฒนาระบบฐานข้อมูลของเรา

คุณสมบัติของ PowerBuilder

- PowerBuilder จัดเป็นภาษาในตระกูล 4GL (Fourth Generation Language) ส่วนนี้ก็คือ PowerBuilder Script ซึ่งเป็นภาษาที่ใกล้เคียงกับภาษามนุษย์ (มนุษย์ที่ใช้ภาษาอังกฤษนะครับ) ถ้าท่านที่เคยใช้ Visual Basic จะรู้สึกคุ้นเคยเพราะมีความคล้ายคลึงกันมากในด้านการไวยากรณ์และคำสั่งพื้นฐาน

- PowerBuilder สนับสนุนการทำงานแบบ Client-Server โดยที่ Sever จะเป็นเพียงฐานข้อมูลที่คอยให้บริการแก่ Client ในด้านการประมวลผลแล้วส่งผลลัพธ์กลับไป Client ซึ่งวิธีนี้ช่วยให้ระบบฐานข้อมูลที่เราสร้างขึ้นมีประสิทธิภาพเพิ่มขึ้นโดยเฉพาะความเร็วในการทำงาน เนื่องจาก Sever ไม่ต้องทำงานทุกอย่างต่างจากระบบการทำงานแบบรวมทุกอย่างไว้ใน Sever พวกเราที่เคยพัฒนาโปรแกรมใน DOS และใช้ NetWare คงคุ้นเคยกับระบบการใช้ Server เป็นศูนย์กลางนี้ดี

- PowerBuilder เน้นการพัฒนาแอพพลิเคชันบน Windows ที่เป็นระบบฐานข้อมูลเชิงสัมพันธ์ (Relational Database Management-RDBMS) โดยการใช้ SQL (Structure Query Language) ในการติดต่อกับระบบฐานข้อมูล

> ☒ Relational ถ้าจะอธิบายง่าย ๆ ก็หมายถึง ภาพที่ยูสเซอร์จะมองเห็นข้อมูลในระบบ โดยที่ข้อมูลจะมาจาก Table ที่มีความสัมพันธ์กัน และข้อมูลที่ยูสเซอร์เห็นไม่ใช่ Table ใหม่ แต่เป็นการดึงข้อมูลใน Table ที่มีความสัมพันธ์มาแสดงเท่านั้น ซึ่งทำให้เกิดภาพที่เป็นความสัมพันธ์เป็นอันหนึ่งอันเดียวกันในฐานข้อมูล (รายละเอียดมีเยอะครับ ถ้ามีโอกาสจะเขียนมาให้อ่านอีกที)
>
> ☒ SQL เป็นภาษาระดับสูงที่ได้รับความนิยมสูงสุด ใช้ในการระบุรายละเอียดของการกระทำ Relational กับ Table ที่อยู่ในระบบฐานข้อมูล มีรากฐานมาจากพีชคณิตแบบ Relational และถูกกำหนดเป็นมาตรฐานในหลายองค์กร (รวมถึงคณะกรรมการของ ANSI) เราคงจะได้กล่าวถึงเรื่องนี้กันอีกเพราะมีความเกี่ยวข้องโดยตรงกับ PowerBuilder

● การพัฒนาระบบงานใน Power Builder จะเป็นในลักษณะ Event/Driven เป็นการเขียนที่มองทุกองค์ประกอบเป็นวัตถุ และวัตถุนั้นจะถูกกระตุ้นให้ทำงานหรือมีคุณสมบัติเปลี่ยนไปตามความเหมาะสม

● PowerBuilder สนับสนุนการเขียนโปรแกรมเชิงวัตถุ (OOP) สามารถสืบทอดคุณสมบัติวัตถุที่มีอยู่แล้วมาใช้ได้อีก ทำให้การแก้ไข/เพิ่มเติมโปรแกรมทำได้ง่าย และร่นระยะเวลาการพัฒนาไปได้มาก นอกจากนั้นยังสนับสนุนการพัฒนาโปรแกรมใช้ได้หลาย Platform โดยแทบไม่ต้องแก้ไข Code

ส่วนประกอบของ PowerBuilder

1. Project Painter เป็นที่เก็บคุณสมบัติของ Application ต่าง ๆ เช่น ชื่อ File Executable, ไอคอนของ Application เป็นต้น นอกจากนั้นยังเป็นที่เก็บชื่อ File พิเศษที่ใช้ในระบบงาน เช่น PowerBuilder Dynamic Library (.PBD) และ PowerBuilder Resource (.PBR)

2. Application Painter เป็นที่รวมของ Object ที่จะใช้ในโปรแกรมทุกชนิด รวมถึง Library search paths, Font สำหรับ Header คอลัมน์ Label, ฟังก์ชันที่ใช้เป็นสาธารณะ (เรียกใช้ได้ทั่วไปทั้ง Application), ตัวแปรสาธารณะ

3. Power Script Painter เป็นภาษาที่ใช้ใน PowerBuilder ซึ่งมีความคล้ายคลึงกับ Visual Basic ใช้ในการสั่งงานให้แต่ละ Object ใน PowerBuilder ทำงานตามความต้องการของผู้พัฒนาระบบ

4. Database Painter เป็นส่วนของการจัดการกับฐานข้อมูล ซึ่งมีหลากหลายตามความต้องการ เช่น การทำงานต่าง ๆ เกี่ยวกับ Database เปลี่ยนแปลงแก้ไข Foreign Key ลบ Index, Foreign keys, Views, table กำหนดคุณสมบัติใน Table เช่น Font สำหรับ table data, heading, label สร้าง/แก้ไข/ลบ Primary Key

5. Data Administration painter เป็นส่วนควบคุมการติดต่อกับฐานข้อมูล สร้างผู้ใช้และกรุ๊ป ส่วนที่เกี่ยวข้องกับการประมวลผล SQL คือ สร้าง นำเข้า และนำข้อความ SQL จากคลิปบอร์ด มาเติม ณ จุดที่กำหนด และประมวลผลประโยค SQL นั้น

6. Data Manipulation Painter เกี่ยวข้องกับ Database คือใช้ในการ แสดง แก้ไข เรียงลำดับ นำเข้า ส่งออก และจัดพิมพ์

7. Datapipeline Painter ใช้ในการก๊อบปี้ข้อมูลจาก Database หนึ่ง ไปยังอีก Database หนึ่ง โดยเราต้องระบุ Data source ของ Database ที่เป็นต้นทาง Database ที่เป็นปลายทาง ตลอดจนเงื่อนไขอื่น ๆ เช่น ตัวแปรที่เกี่ยวข้อง ลักษณะการรวมกัน การจัดเรียง การคัดเลือก การรวมกลุ่มเป็นกรุ๊ป เป็นต้น

8. Query painter ใช้ในการสร้าง/แก้ไข และเก็บประโยค SQL นำไปใช้ใน Data Window Object

9. Report Painter สร้าง Report ในรูปแบบต่าง ๆ ทั้งเป็นการแสดงข้อมูลแบบ Text และเป็นกราฟิก เช่น กราฟต่าง ๆ

10. Library Painter ใช้สำหรับจัดการกับ Library Files ไม่ว่าจะเป็นการเรียกใช้ แก้ไข สร้างขึ้นใหม่ หรือ Inherit Object มาเป็น Object ใหม่

11. DataWindow Painter ซึ่งใช้ในการกำหนดความสัมพันธ์ระหว่าง Table (ในรูปของ Select SQL), การสร้าง/แก้ไข SQL, สร้าง Store procedure เพื่อใช้ร่วมกับ DBMS ของ PowerBuilder ตลอดจนนำข้อมูลมาแสดงในรูปแบบต่าง ๆ ไม่ว่าจะเป็นการเพิ่มเติม/แก้ไข/ลบข้อมูล การนำข้อมูลมาแสดงในรูปแบบของตาราง กราฟชนิดต่าง ๆ โดยสามารถใช้รูปแบบผสมผสานกันได้ตามความต้องการของผู้พัฒนาโปรแกรม

12. Menu Painter ใช้สำหรับสร้างเมนูและ Toolbars

13. Function Painter สร้างฟังก์ชันประเภท User Define Function ขึ้นใช้ตามความต้องการ

14. File Editor ใช้แก้ไข Text file เช่น Source (*.sr*), Resource (*.pbr), Initialization (*.ini), Text (*.txt)

15. Select Painter ใช้เมื่อมีการใช้ SQL select datasource (จะพบในการสร้าง Datawindow Object) เราสามารถเลือกเปิดแก้ไข กำหนดความสัมพันธ์ระหว่างตาราง สร้างการรับค่าตัวแปร คัดเลือกข้อมูลที่ต้องการ จัดเรียงข้อมูล เป็นต้น โดยเก็บเป็น SQL query

16. Structure Painter สำหรับสร้าง Structure (กลุ่มของตัวแปร) Structure ของ PowerBuilder นี้ใน Delphi หรือ Visual Basic จะเรียกว่า Record ซึ่งก็แตกต่างกันไปตามการบัญญัติศัพท์

17. Table Painter ใช้สำหรับสร้าง เปลี่ยนแปลง และกำหนดคุณสมบัติของฟิลด์ใน Table เช่น Column Header, Format, Validation, รูปแบบการแสดงข้อมูล

18. User Object Painter สำหรับสร้าง Object ขึ้นมาเองตามความต้องการ ซึ่งเราสามารถเรียกมาใช้ใน Window Object ได้

19. View Painter สร้างการแสดงข้อมูลเราสามารถดึงมาใช้ใน Datawindow Object ได้ (พวกเราที่เคยใช้ FoxPro คงคุ้นเคยกับการสร้าง View แล้วเรียกใช้โดย Set Format to..... ซึ่งก็คล้าย ๆ กัน)

20. Window Painter ใช้สำหรับสร้าง Window เชื่อมโยงกับเมนู ดึง Window ย่อยมาประกอบ และปรับแสดงคุณสมบัติต่าง ๆ ของ Window

Toolbar

PowerBuilder มีแถบเครื่องมือที่ช่วยในการทำงานสร้าง Application แถบเครื่องมือหลักคือ PowerBar เป็นแถบบนสุดที่อยู่ใต้เมนู (รูปที่ 1) ไว้อำนวยความสะดวกในสร้าง Object ต่าง ๆ ซึ่งในหน้าจอการสร้าง Object เหล่านี้จะมีแถบเครื่องมือเป็นของตัวเองอีกอยู่ถัดจาก Powerbar ลงมา (โปรแกรมแบบ Visual ทั่วไปมักมี Toolbar ในลักษณะนี้ รวมถึงลักษณะของเมนูด้วย)

มีส่วนประกอบมาก แต่ไม่ได้หมายความว่าการใช้โปรแกรมจะยาก

พวกเราบางคนอ่านมาถึงตรงนี้ คงรู้สึกว่าเครื่องมือใน PowerBuilder ช่างมีมากมาย การใช้งานจริงคงสับสนอลหม่านพิลึก ไม่เป็นถึงขนาดนั้นหรอกครับ เพราะ PowerBuilder ได้เตรียมส่วนนี้ไว้ก็เพื่อแยกงานแต่ละกลุ่มออกจากกัน สะดวกในการจดจำและเรียกใช้ให้เหมาะกับความประสงค์ของเรา และตามความเป็นจริงเราก็ไม่จำเป็นต้องเรียกใช้ไปทุกส่วนประกอบ คงมีไม่กี่ส่วนประกอบที่เราต้องใช้ประจำ ก็ค่อย ๆ รู้จักกันไป พอชินแล้วทุกอย่างก็จะดูง่ายไปเอง

Application Object

ต่อไปนี้เราจะมาสร้าง Application แรกใน PowerBuilder อยากให้พวกเราลองทายดูว่า การพัฒนา Application แรกให้ทำงานใน PowerBuilder เราจะต้องเขียนโค้ดกี่บรรทัด ให้เวลา 5 วินาทีครับ หมดเวลา...คุณตอบมากี่บรรทัด 5...10....15.....20 ผิดครับ! ที่ถูกต้องคือเราไม่ต้องเขียนโค้ดแม้แต่บรรทัดเดียว ทั้งนี้เพราะ Application แรกของเรา เราจะใช้ตัว Generate ของ PowerBuider ช่วยทำให้การเตรียมโครงสร้างพื้นฐานสำเร็จไปโดยไม่ต้องออกแรงเลย

เมื่อเข้ามาสู่ PowerBuilder แล้ว ให้เราเข้าใน Window Painter เลือก Menu File>New เพื่อสร้าง Application ใหม่ของเรา ในที่นี้จะใช้ชื่อว่า example1 เมื่อเราพิมพ์ชื่อ Application เสร็จแล้วจะขึ้นไดอะล็อกบ็อกซ์ถามว่าต้องการใช้ตัว Generate ของ PowerBuilder หรือไม่ เราจะคลิก Yes แล้ว Application แรกของเราก็สำเร็จ (ยังไม่ได้เขียนโปรแกรมเลยสักบรรทัด)

ขั้นนี้ถ้าเป็นนักพัฒนาที่มีประสบการณ์ก็จะใช้ความชำนาญในการปรับแต่งและเพิ่มเติมจนกลายเป็น Application ที่สมบูรณ์ตามความต้องการ แต่ในตอนนี้เราจะยังไม่ผ่านไป เพราะนี่เป็นโอกาสที่เราจะได้รู้จักกับลักษณะการทำงานของ PowerBuilder ตั้งแต่ขั้นพื้นฐาน ไม่จำเป็นต้องรีบร้อนครับ เพราะถ้าพื้นฐานไม่ดี แต่เขียนตะลุยไปโดยไม่ประมาณตัวมีโอกาสตกจากที่สูงหรือติดค้างอยู่บนนั้นได้ง่าย ถึงตอนนั้นก็สายเกินไปที่จะแก้ตัว

Example1 ที่ถูก Generate มามีโครงสร้างดังนี้ครับ (รูปที่ 2)

ลำดับชั้นของ Object ที่แสดงใน Application Painter ไม่ใช่เกิดขึ้นลอย ๆ แต่แฝงไว้ด้วยความหมายซึ่งทำให้เรามองภาพรวมของ Application ได้อย่างชัดเจน นั่นคือ

ตัวหลักเริ่มแรก example1 เป็น object Application ซึ่งจะเรียกใช้ Object window ตัวรองลงมาคือ w_genapp_frame ใน w_genapp_frame จะมี Object ย่อยแยกเป็นสองสายคือ

1. สายของ เมนู m_genapp_frame ซึ่งจะควบคุมไปถึง Windows 2 ตัวคือ w_genapp_toolbars และ w_genapp_about

2. สายของ Window w_genapp_sheet โปรดสังเกตว่า เมนู m_genapp_frame เมื่ออยู่ในสถานะของ Windows w_genapp_sheet แอคทีฟอยู่จะถูกลดระดับเป็นเมนูระดับสอง โดยเมนูที่ถูกเรียกใช้คือ m_genapp_sheet โดยที่ windows สองตัวก็คือ w_genapp_toolbars และ w_genapp_about ยังสามารถเรียกใช้งานได้อยู่เพียงแต่ถูกลดความสำคัญลงไป

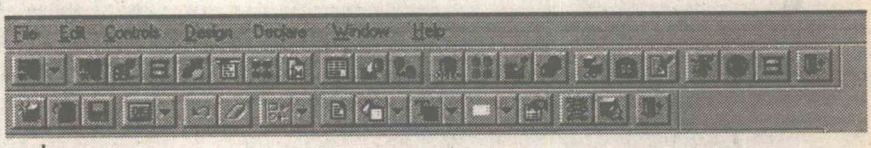
กฎแห่งศักดินา?

เรื่องของการแบ่งชนชั้นยังเข้มข้นสำหรับในเรื่องของ Object เพราะมันเป็นเรื่องความแตกต่างของวงศ์ตระกูล และศักดิ์ศรีซึ่งผูกติดกับการสนิทชิดเชื้อกับผู้มีอำนาจบาตรใหญ่

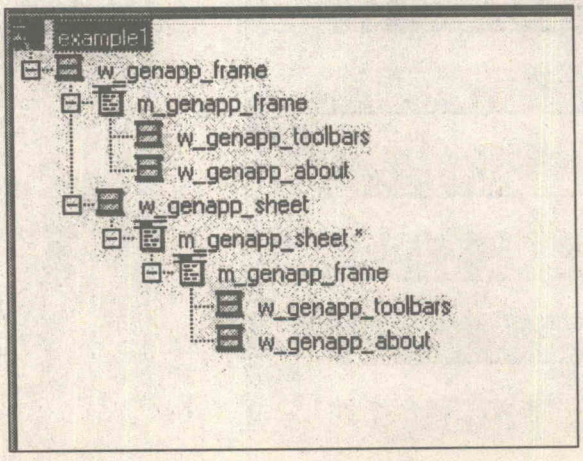
วงศ์ตระกูล หมายถึงต้นกำเนิดที่ตกทอดกันมาแต่บรรพบุรุษ เช่น Window ก็มีหลายชนิด แต่ละชนิดก็มีชนชั้นแตกต่างกันไป เช่น MDI Frame ย่อมเหนือ Main และ Main ย่อมเหนือกว่า Respond

ศักดิ์ศรี ก็มีส่วนสำคัญในการแบ่ง แม้จะเป็นเมนูเหมือนกัน แต่เมนูที่ผูกติดกับ Window ที่แอคทีฟอยู่ขณะนั้นย่อมมีชนชั้นเหนือกว่าเมนูตัวอื่น ๆ

ถ้าอ่านแล้วยังสับสน เราคงจะต้องไปดูกันว่า Object ที่เห็นอยู่นี้แต่ละตัวมีคุณสมบัติอย่างไร และทำงานอย่างไร



รูปที่ 1



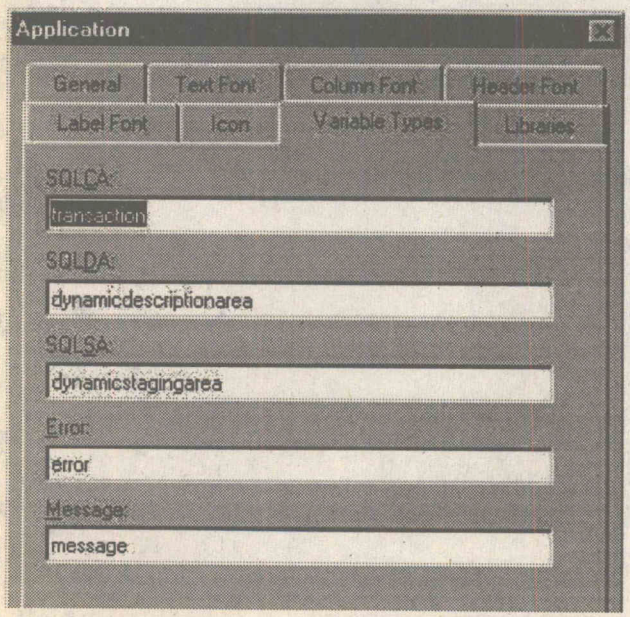
รูปที่ 2

```
01  /*****************************************************
02  Application Open Script
03  Selects start-up file according to Operating System
04  Populates sqlca from start-up file
05  Connects to DB (if uncommented)
06  Opens frame
07  */
08  environment        env              // holds environment information
09  string     startupfile              // holds name of start-up file
10  /* Get the environment information */
11  IF ( GetEnvironment(env) <> 1 ) THEN
12  MessageBox( "Application: Open", "Unable to get environment information.~nHalting
...")
13  HALT
14  END IF
15  /* Select start-up file by operating system */
16  CHOOSE CASE env.OSType
17  CASE Windows!, WindowsNT!
18  startupfile = "pb.ini"
19  CASE Sol2!, AIX!, OSF1!, HPUX!
20  startupfile = ".pb.ini"
21  CASE Macintosh!
22  startupfile = "PowerBuilder Preferences"
23  CASE ELSE
24  MessageBox( "Application: Open", "Unrecognized operating system.~nHalting ..." )
25  HALT
26  END CHOOSE
27  /* Populate sqlca from current PB.INI settings */
28  sqlca.DBMS       = ProfileString (startupfile, "database", "dbms",       "")
29  sqlca.database   = ProfileString (startupfile, "database", "database",   "")
30  sqlca.userid     = ProfileString (startupfile, "database", "userid",     "")
31  sqlca.dbpass     = ProfileString (startupfile, "database", "dbpass",     "")
32  sqlca.logid      = ProfileString (startupfile, "database", "logid",      "")
33  sqlca.logpass    = ProfileString (startupfile, "database", "LogPassWord", "")
34  sqlca.servername = ProfileString (startupfile, "database", "servername", "")
35  sqlca.dbparm     = ProfileString (startupfile, "database", "dbparm",     "")
36  /* Uncomment the following for actual DB connection */
37  //connect;
38  //
39  //if sqlca.sqlcode <> 0 then
40  // MessageBox ("Cannot Connect to Database", sqlca.sqlerrtext)
41  // return
42  //end if
43  /* Open MDI frame window */
44  Open (w_genapp_frame)
```

ลิสติงที่ 1

อธิบายเพิ่มเติม ลิสติงที่ 1

โปรแกรมจะเริ่มทำงานโดยการตรวจเช็คสภาพแวดล้อม และค่าเริ่มต้นของระบบ บรรทัดที่ 08-09 กำหนดตัวแปร Env ให้เป็น Object ชนิด Environment และ Startup เป็นตัวแปร String ตัวแปรทั้งสองตัวจะนำไปใช้ในการตรวจเช็คสภาพแวดล้อม นั่นคือ ชนิดของ OS และชื่อไฟล์ Startup ในบรรทัดที่ 11-26 ต่อจากนั้นคือการเตรียมงานสำหรับการติดต่อ Database ที่โปรแกรมรู้จัก (บรรทัด 28-41) แล้วจึงมาเปิด Window หลักที่ชื่อ w_genapp_frame ในบรรทัดสุดท้าย ถึงตอนนี้ Window w_ genapp_frame ก็พร้อมที่จะทำงานแล้ว



รูปที่ 3

Example 1

เป็น object Application เราสามารถเข้าไปดูคุณสมบัติได้ใน Properties (รูปที่ 3)

ส่วนนี้ที่เราควรเข้าไปปรับแต่ง เช่น การแสดงของ Font ใน Application ให้เป็นภาษาไทย (ปกติจะเป็น Arial) การเปลี่ยน Icon ของ Application เป็นต้น

ส่วนที่สำคัญใน Application คือส่วนของ Script ใน Event Open ซึ่งจะถูกเรียกใช้เป็นอันดับแรกในการเปิด Application ซึ่งจะมีรูปแบบดังลิสติงที่ 1

Window w_genapp_frame

มี Properties ที่น่าสนใจดังตารางที่ 1

Event ที่เกี่ยวข้อง

ส่วนที่อาจทำความสับสนคือเมื่อเราเข้าไปเปิดดู Script บางครั้งเราจะพบว่ามีการเรียกใช้ฟังก์ชันชื่อแปลก ๆ ซึ่งเปิดดูใน Help ไม่เจอ ครั้งแรกผู้เขียนก็ปวดหัวเช่นกัน จนกระทั่งไปเปิดเจอในเมนู Declare >Window Functions... จึงถึงบางอ้อ ที่แท้ฟังก์ชันพวกนี้

เป็นฟังก์ชันประเภท User Define Function นั่นเอง ใน window w_genapp_fram Event Open ของ Script เป็นดังนี้

```
/* Modify men/u text for platform */
wf_setmenutext (menuid)
/* Create the initial sheet */
wf_newsheet ()
```

เราจะเห็นว่ามีฟังก์ชันชื่อแปลก ๆ ถึงสองตัว ฟังก์ชันทั้งสองนี้เป็นฟังก์ชันที่เขียนขึ้นใช้ในเฉพาะ Window w_genapp_frame เท่านั้น ซึ่งก็มีรายชื่อ รหัสโค้ดและหน้าที่การทำงานดังนี้ (ดูรูปที่ 4 ประกอบ)

ฟังก์ชัน wf_closesheet()

```
window lw_Sheet
   /* Close active sheet */
   lw_Sheet = this.GetActiveSheet ()
   if IsValid (lw_Sheet) then
      Close (lw_Sheet)
  end if
```

มีหน้าที่ : ปิด Window w_genapp_sheet ที่เปิดอยู่ในปัจจุบัน

ฟังก์ชัน wf_enableprint(boolean enable)

```
/* Enable or disable printing */
if enable then
   TriggerEvent (*enableprint*, 1, 0)
else
   PostEvent (*enableprint*, 0, 0)
end if
```

มีหน้าที่ : อนุญาตหรือไม่อนุญาตให้มีการพิมพ์ โดยมีพารามิเตอร์คือ enable เป็นชนิด Boolean

ฟังก์ชัน wf_getsheetcount()

```
int      li_Count
window lw_Sheet
   /* Return number of open sheets */
   lw_Sheet = this.GetFirstSheet ()
   if IsValid (lw_Sheet) then
      do
         li_Count += 1
         lw_Sheet  = this.GetNextSheet
(lw_Sheet)
      loop while IsValid (lw_Sheet)
   end if
  return li_count
```

มีหน้าที่ : นับจำนวน w_genapp_sheet ที่เปิดอยู่ทั้งหมด ส่งค่ากลับเป็น integer

ฟังก์ชัน wf_newsheet()

```
w_genapp_sheet lw_Sheet
   /* Enable printing */
   wf_enableprint (true)
   /* Open a new instance of a sheet */
  return OpenSheet (lw_Sheet, this, 3,
layered!)
```

มีหน้าที่ : เปิด w_genapp_sheet เพิ่ม ส่งค่าจำนวน Sheet ล่าสุดกลับเป็น Integer

ฟังก์ชัน wf_setmenutext (m_genapp_frame menu_reference)

```
Environment lenv_env
   /* Change menu "Exit" text based on
platform */
   GetEnvironment (lenv_env)
   if lenv_env.ostype = Windows! then
      menu_reference.m_file.m_exit.text
= "E&xit"
   elseif lenv_env.ostype = Macintosh!
then
      menu_reference.m_file.m_exit.text
= "Quit"
  end if
```

มีหน้าที่ : Set text ในเมนู m_genapp_frame หัวข้อ File (ชื่อ M.file) ชื่อ Exttext โดยจำแนกตาม OS ถ้า OS เป็น Windows จะใช้คำว่า Exit ถ้าเป็น Macintosh จะใช้ Quit โดยมีพารามิเตอร์ คือ menu_ reference เป็น object ชนิดเมนู สืบทอดมาจาก m_genapp_ frame

ถ้าส่วนนี้ยังอ่านไม่เข้าใจก็ไม่เป็นไรนะครับ โดยเฉพาะผู้ไม่มีพื้นความรู้ภาษา Visual Basic คงลำบากหน่อย ถ้าเรามีความสนใจภาษา Power Script ศึกษาจนเกิดความเข้าใจในระดับหนึ่ง รับรองว่าการเขียนโค้ดใน PowerBuilder จะไม่ใช่เรื่องยากอีกต่อไป ในโอกาสหน้าผู้เขียนจะค่อย ๆ สอดแทรกข้อมูลเกี่ยวกับใช้คำสั่งไปเรื่อย ๆ จนกว่าผู้อ่านจะเกิดความเข้าใจมากขึ้น

กลับมาดู ส่วน Event Open ของ w_genapp_frame กันต่อครับ จะเห็นว่าก็เป็นการใช้ฟังก์ชันที่ประกาศไว้ใน Windows Functions... นั่นเอง กล่าวคือ

1. Set ค่าText ในเมนูให้ตรงกับ OS
2. เปิด w_genapp_sheet ขึ้นใหม่

Event Enableprint

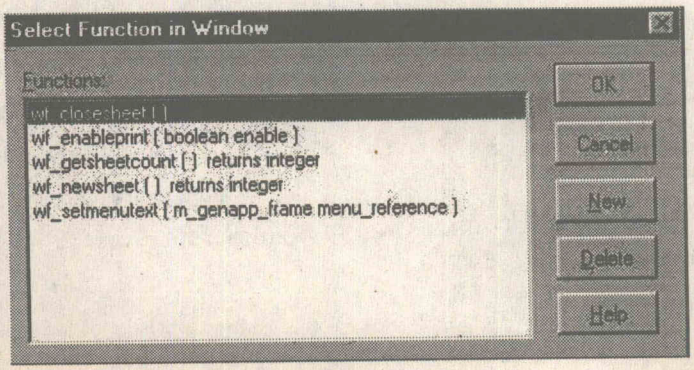
```
boolean lb_enable
m_genapp_frame lm_genapp_frame
   /* Enable or disable printing */
   lm_genapp_frame = menuid
   if message.wordparm <> 0 then
lb_enable = true
   lm_genapp_frame.m_file.m_print.enabled
= lb_enable
```

Event นี้มีหน้าที่ตรวจเช็คว่ามี w_app_sheet เปิดอยู่ในระบบหรือไม่ ถ้ามีอย่างน้อย 1 ตัวจึงอนุญาตให้มีการพิมพ์ได้ ถ้าไม่จะไป off ที่เมนูย่อย Print ของ m_genapp_frame.m_file

ตารางที่ 1

Tab	Name	Value
General	Title	Frame
	Menuname	M_genapp_frame
	Window type	MDI Frame with Microhelp
	Window color	AppWrkSpc
	MDI client color	AppWekSpc
Position	Intitial State	Maximize

รูปที่ 4



Window w_genapp_sheet

เป็น Window ที่ถูกเรียกใช้จาก w_genapp_frame ในลักษณะเป็น Sheet คล้ายกับการเปิด Sheet ใหม่ของ Excel มีลำดับเป็น Sheet1, Sheet2, ... มี Properties ที่น่าสนใจดังตารางที่ 2

Event Open

```
int li_Count
   /* Sheet opening - reflect sheet count
in title */
   li_Count =
w_genapp_frame.wf_getsheetcount ()
   this.Title = "Sheet:" + string
(li_Count)
   /* Modify menu text for platform */
   w_genapp_frame.wf_setmenutext (menuid)
```

Event ตัวนี้มีลักษณะที่น่าสนใจอย่างหนึ่งก็คือ มีการเรียกใช้ฟังก์ชันที่อยู่ Window ตัวอื่น คือ เรียกใช้ w_genapp_frame.wf_ getsheetcount() และ w_genapp_frame.wf_ setmenutext (menuid) ที่เป็นเช่นนี้เพราะ Object ทั้งสองความเกี่ยวข้องกันค่อนข้างใกล้ชิด (ห่างไม่เกิน 2 ชั้น) นอกจากนั้นต่างก็เป็น Object ชนิด Window ซึ่งมีคุณสมบัติที่คล้ายคลึงกัน สามารถอ้างอิงได้โดยตรง (การควบคุมเมนูก็มีผลมาจากความเกี่ยวข้องของ Object ที่มีชั้นห่างกันไม่มากนัก)

เมื่อ w_genapp_sheet ถูกเปิดขึ้น Event open จะทำงานโดยการนับจำนวน sheet ที่เปิดอยู่ทั้งหมด รวมทั้งตัวมันด้วยแล้วเติมเลขที่นับได้ไปท้าย Title

จุดอ่อน?

การทำงานตรงส่วนนี้ยังมีจุดอ่อนกล่าวคือ สมมติว่าเราสร้าง sheet ใหม่ทั้งหมด 4 sheet คือ sheet1, sheet2, sheet3, sheet4 แล้วเราปิด sheet3 เมื่อเราสร้าง sheet ใหม่ขึ้นอีก 1 sheet ข้อผิดพลาดที่เกิดขึ้นคือ sheet4 จะมีซ้ำเป็น 2 ชุด ทั้งนี้เพราะโปรแกรมจะนับตามจำนวน sheet ที่มีอยู่จริง โดยไม่ได้สนใจว่าชื่อ sheet จะซ้ำกันได้หรือไม่ ซึ่งถ้าเป็นพัฒนาโปรแกรมเพื่อใช้งานจริงคงต้องเพิ่มส่วนป้องกันความผิดพลาดที่เกิดขึ้นด้วย

Event Close

```
/* Disable printing if last sheet */
if w_genapp_frame.wf_getsheetcount () = 1
then
   w_genapp_frame.wf_enableprint (false)
end if
```

Event นี้มีหน้าที่ตรวจเช็ค คือถ้า sheet ที่จะปิดนี้เป็น sheet สุดท้ายที่เหลืออยู่แล้ว มันจะไปปิดการทำงานในส่วนของการพิมพ์ (ไม่มีข้อมูล-ไม่รู้จะพิมพ์อะไร?)

M_genapp_frame และ M_genapp_sheet

ก็มาถึงส่วนของเมนูแล้วครับ การสร้างเมนูของ PoweBuilder ก็มีส่วนคล้ายกับเครื่องมือพัฒนาตระกูล Visual อื่น โดยเน้นที่ความง่ายต่อการสร้างและเรียกใช้งาน (ดูรูปที่ 5) แต่มีจุดเด่นอย่างหนึ่งที่เครื่องมือตัวอื่นไม่มีคือการสามารถสร้างเมนูไปพร้อมกับ Toolbar ได้ เพราะทั้งสองอย่างนี้มีคุณประโยชน์คล้ายกัน ดังนั้นเราจึงไม่จำเป็นต้องไปแยกสร้างแล้วเรียกใช้ อย่างโปรแกรม Visual ตัวอื่น ๆ จึงถือเป็นลักษณะพิเศษที่ไม่น่ามองข้าม

เมนูทั้งสองมีลักษณะการสร้างและการทำงานที่คล้ายคลึงกัน เพียงแต่ถูกเรียกใช้โดย window คนละตัวเท่านั้น

สำหรับเรื่องของเมนู มีรายละเอียดปลีกย่อยอีกมาก คงได้มีโอกาสกล่าวถึงในลักษณะเจาะลึกในบทความตอนต่อ ๆ ไป

w_genapp_abount

สำหรับ window ตัวนี้ไม่มีอะไรที่ซับซ้อนเกินความเข้าใจ การทำงานก็คือเป็นตัวแสดงข้อมูลเกี่ยวกับ Application แต่เพียงย่อ ๆ และยูสเซอร์มีหน้าที่เพียงกด OK เพื่อออกมาสู่ window หลักเท่านั้น การปรับแต่งคือเราสามารถพิมพ์ข้อความลงไปตามต้องการ เนื่องจากเกรงว่าบทความนี้จะยาวเกินไปผู้เขียนจึงขอข้ามไม่กล่าวถึงในรายละเอียด

w_genapp_toolbar

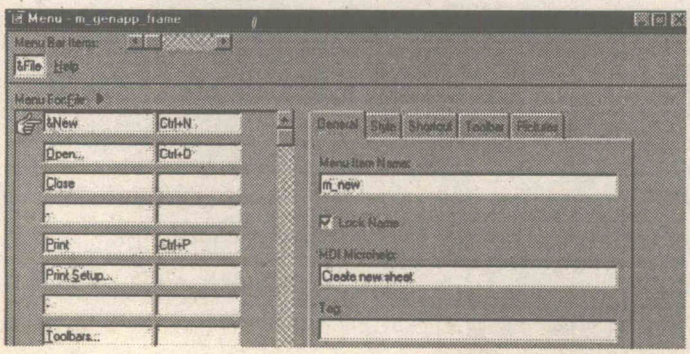
Window ตัวนี้มีหน้าที่ควบคุมการแสดงผลของ Toolbar โดยตรง น่าสนใจครับ (รูปที่ 6) มี Properties ที่น่าสนใจดังตารางที่ 3

ใน w_genapp_toolbar จะมีการใช้ตัวแปรประเภท Instance

ตารางที่ 2

Tab	Name	Value
General	Title	Sheet
	Menuname	M_genapp_sheet
	Window type	Main
	Window color	WndBkrnd
	MDI client color	AppWekSpc
	Maximize Box	Off
	Minimize Box	Off
	Resizeable	Off
Position	Intitial State	Normal

รูปที่ 5



PowerBuilder 5.0

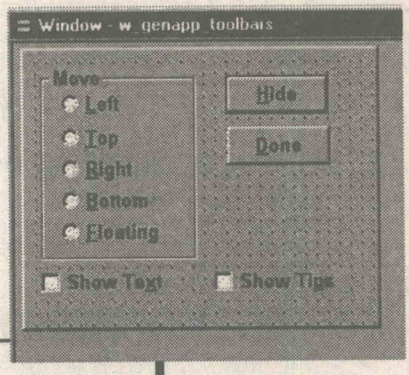
: มุมมองที่แตกต่าง ตอนที่ 2

ลิสติงที่ 2 Event open

```
01  /* Owning toolbar window is passed as reference */
02  iw_window = message.powerobjectparm
03  /* Note the current application */
04  iapp_application = GetApplication ()
05  /* Set toolbar alignment status */
06  choose case iw_window.toolbaralignment
07  case alignatbottom!
08  rb_bottom.checked = true
09  case alignatleft!
10  rb_left.checked = true
11  case alignatright!
12  rb_right.checked = true
13  case alignattop!
14  rb_top.checked = true
15  case floating!
16  rb_floating.checked = true
17  end choose
18  /* Set toolbar visible status */
19  if iw_window.toolbarvisible then
20  cb_visible.text = "&Hide"
21  else
22  cb_visible.text = "&Show"
23  end if
24  /* Set toolbar text mode status */
25  cbx_showtext.checked = iapp_application.toolbartext
26  /* Set toolbar tips mode status */
27  cbx_showtips.checked = iapp_application.toolbartips
```

Tab	Name	Value
General	Title	Toolbars
	Window type	Respond
	Window color	Silver
	MDI client color	AppWekSpc
	Maximize Box	Off
	Minimize Box	Off
	Resizable	Off
Position	Intitial State	Normal

ตารางที่ 2



รูปที่ 6

คำอธิบายเพิ่มเติมลิสติงที่ 2

เริ่มแรกจะเป็นตรวจเช็คสถานะของ Toolbars ในระบบว่าสามารถใช้งานได้ตามปกติหรือไม่ โดยส่งค่าผ่านมาจาก message. powerobjectparm สู่ iw_window ซึ่งเป็น Stance Variable มีชนิดเป็น Window จากนั้นจึงเป็นการเช็คสถานะของ Toolbar ที่แสดงในปัจจุบันจากคุณสมบัติ iw_window. toolba- ralignment ซึ่งจะมีผลกับสถานะของกลุ่ม Object ปุ่ม option ในหัวข้อ Move ของ w_genapp_toolbar ซึ่งจะมีการเช็คปุ่มที่มีค่าตรงกับสถานะของ Toolbars ปัจจุบัน จากนั้นจึงมาเช็คคุณสมบัติ Visible ของ Toolbar ว่าเป็นจริงหรือไม่ จะมีผลกับคุณสมบัติ Text ของ command button cb_visible ที่เราเห็นอยู่คือ command button ที่มี Text ว่า Hide นั่นเอง และจากนั้นจึงเป็นเช็คสถานะของการแสดง Text และ Show Tool tips โดยค่าที่เช็คจะนำไปแสดงที่ปุ่ม Option ในหัวข้อตามระบุ

และเพื่อให้ยูสเซอร์สามารถเลือกค่าการแสดง Toolbars ได้โดยตรง ในแต่ละ Option จึงต้องมี Script ในส่วนของ Event Click ดังนี้

```
rb_bottom
iw_window.toolbaralignment = alignatbottom!
rb_left
iw_window.toolbaralignment = alignatleft!
rb_right
w_window.toolbaralignment = alignatright!
rb_top
iw_window.toolbaralignment = alignattop!
rb_floating
iw_window.toolbaralignment = floating!
cb_visible
if this.text = "&Hide" then
   iw_window.toolbarvisible = False
   this.text = "&Show"
else
   iw_window.toolbarvisible = True
   this.text = "&Hide"
end if
cbx_showtext
iapp_application.toolbartext = this.checked
cbx_showtips
iapp_application.toolbartips = this.checked
cb_2 (Done)
close(parent)
```

Declare> Instance Variables...

```
/* Current application */
application    iapp_application
* Owning toolbar window */
window         iw_window
```

ส่งท้าย

การอ่านบทความนี้สิ่งที่ช่วยเสริมความเข้าใจได้ดียิ่งขึ้น คือการลอง Run Application ดูไปพร้อม ๆ กันด้วย เพื่อจะได้รู้ถึงคุณสมบัติและความเกี่ยวเนื่องของ Object ตลอดจนการทำงานของ Script ในส่วนต่าง ๆ อย่างกระจ่างแจ้ง เป็นส่วนแรกที่เราจำเป็นต้องรู้และทำความเข้าใจก่อนจะลงลึกในส่วนของ object ต่าง ๆ

สำหรับท่านที่มีข้อเสนอแนะหรือมีข้อสงสัยสามารถติดต่อผู้เขียนได้โดยตรงตาม E-mail ข้างล่างนี้

phasitphattaramoon@thaimail.com
jokeuse@health2.moph.co.th